Locating JTAG pins automatically

Hunz (hunz at hunz.org)

talk from ph-neutral 0x7d6

sourcecode and further information:
http://www.c3a.de/wiki/index.php/JTAG_Finder

What is JTAG?
What can you do with it?

- Joint Test Action Group
- Designed to test if all signals are connected properly – especially for BGA ICs
- All connections of the chip can be read and written -> Access to all connected devices (Flash, etc.)
- One can read/write the Flash without soldering
- Debug Interface: Allows rw-access to RAM and registers! (+Breakpoints)

HDTV Receiver

Where can you find JTAG?

- In nearly every embedded device (PDA, mobile phone, set-top-box)

JTAG-Access isn't very fast

- Used to get the Bootloader into the device
- Downloading the complete Flash via JTAG can take quite a while
- Understanding all the disassembled code might be quite ugly (Thumb/32bit Mode for ARM, etc.)
- Use the JTAG in combination with other methods
  - -> Download only necessary parts (Bootloader) or
  - -> Upload / modify code to dump the memory on a faster interface (RS-232, USB, etc.)
  - Bus sniffing, RS-232 Access

How does JTAG work?

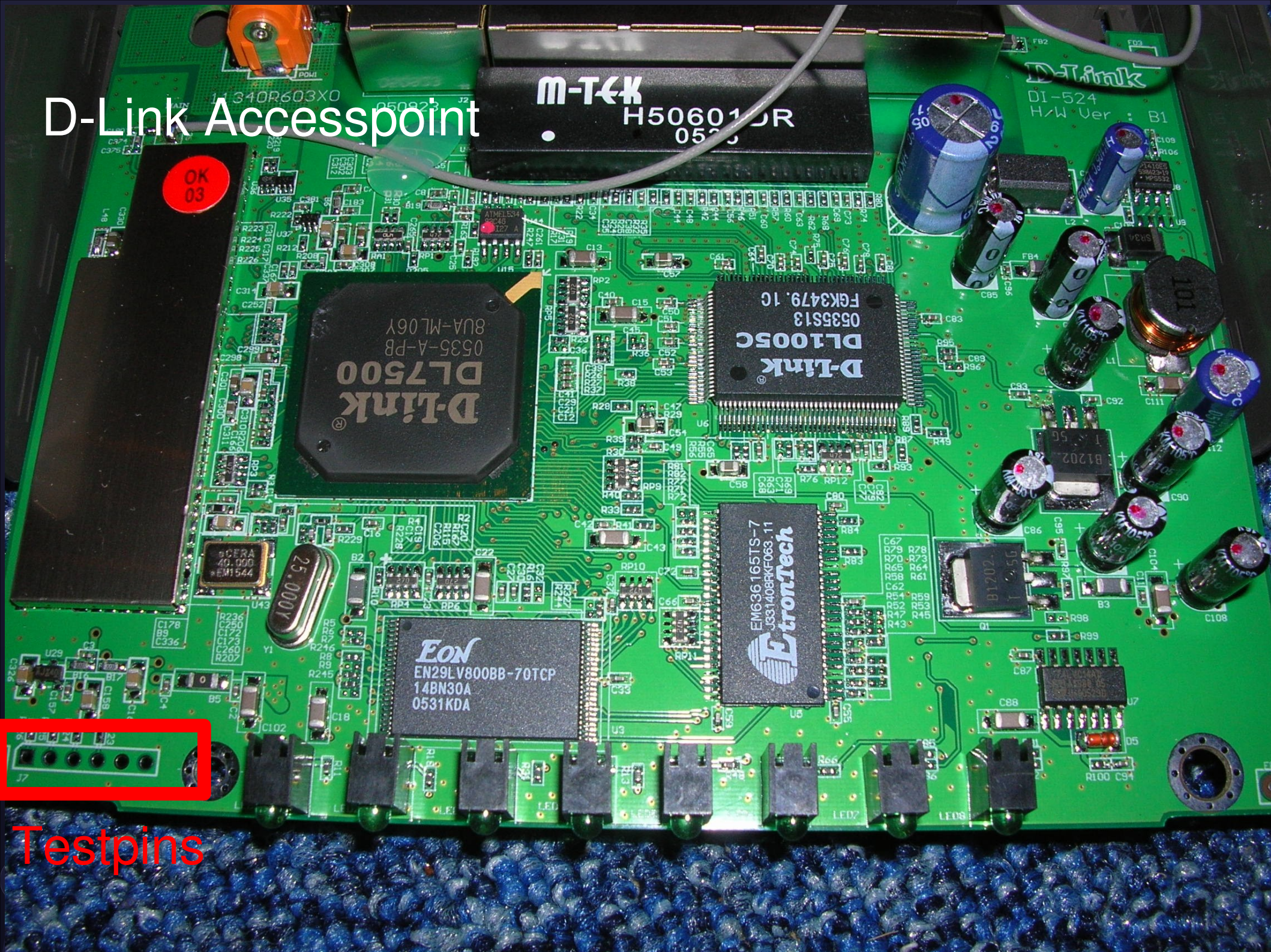
- synchronos serial interface
- 4 or 5 pins:
  - TCK: Test Clock
  - TMS: State Machine Control
  - TDI: Test Data In
  - TDO: Test Data Out
  - (nTRST: Test Reset)
- There are shift registers for Instructions and Data
- and a state machine

Interfacing the JTAG pins

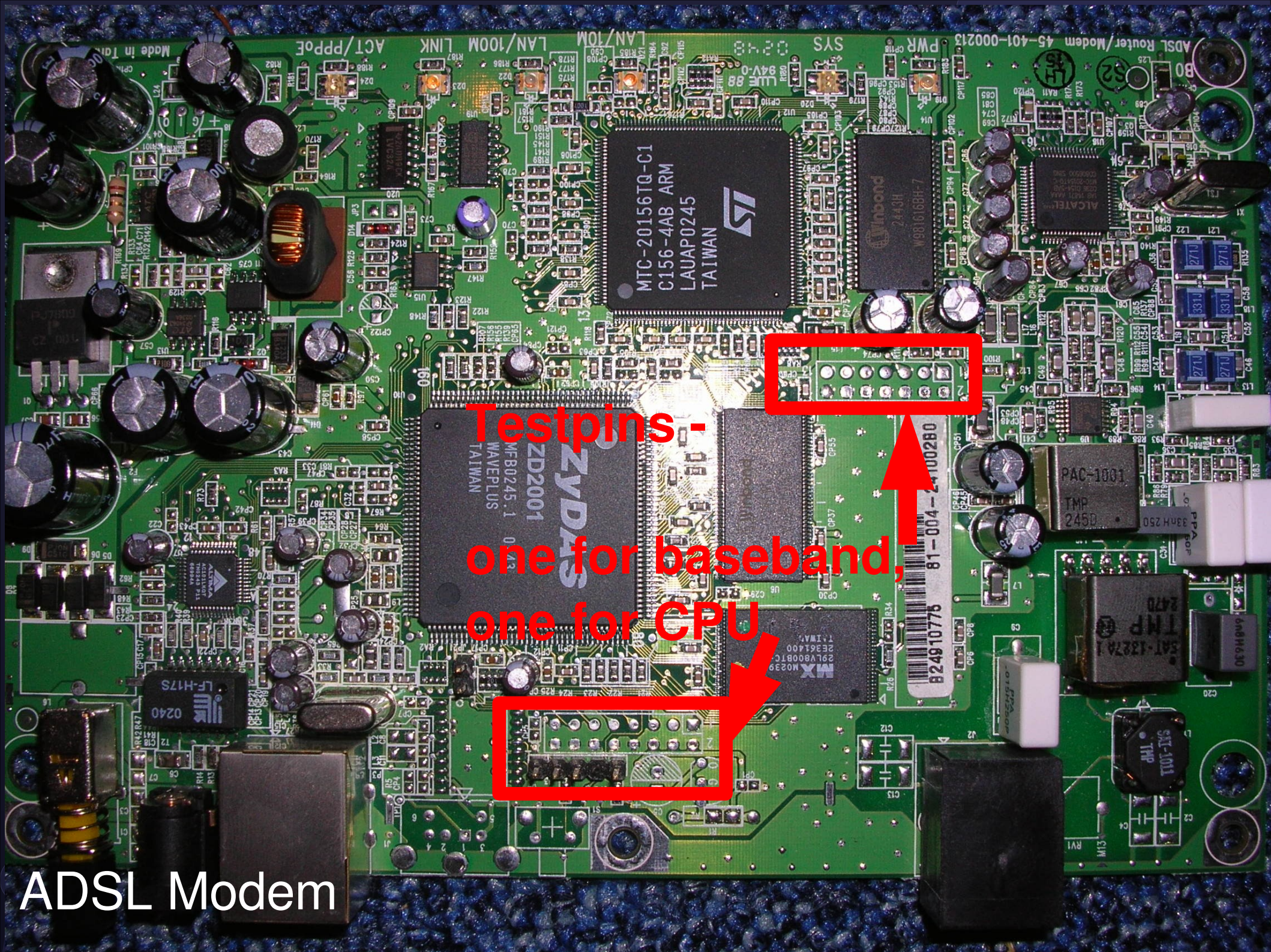
- Parallel port cable (DLC5, Wiggler, etc.)
  - Bit-Banging (Shifting is done by software)
- USB, etc.
  - Hardware shifting: Better speed for large data reads/writes
- Software: There's the openwince-jtag package
  - It does support most parallel port cables
  - It's Opensource
  - There's already bus-access and Flash reading/writing support

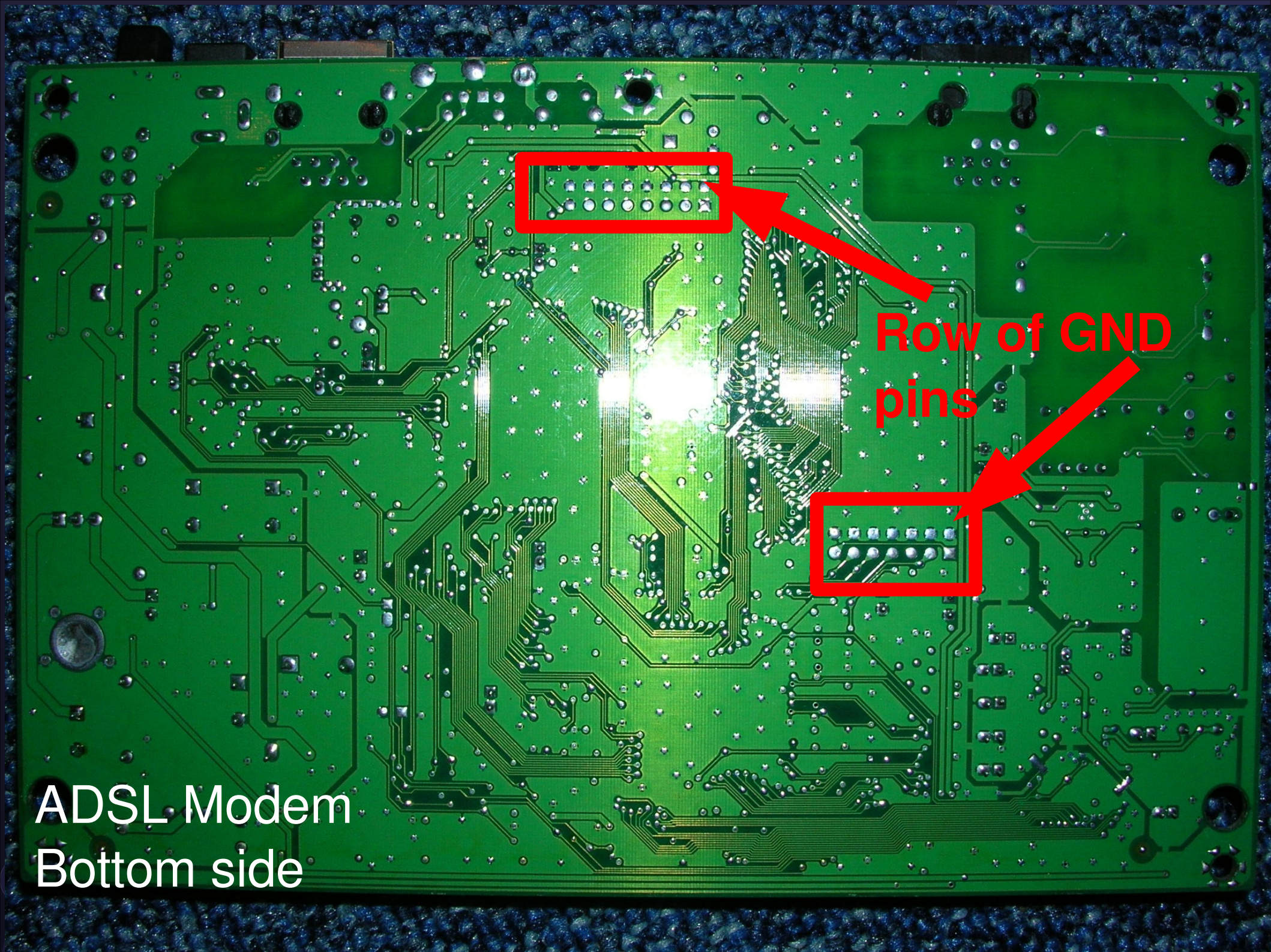
Locating the pins by hand

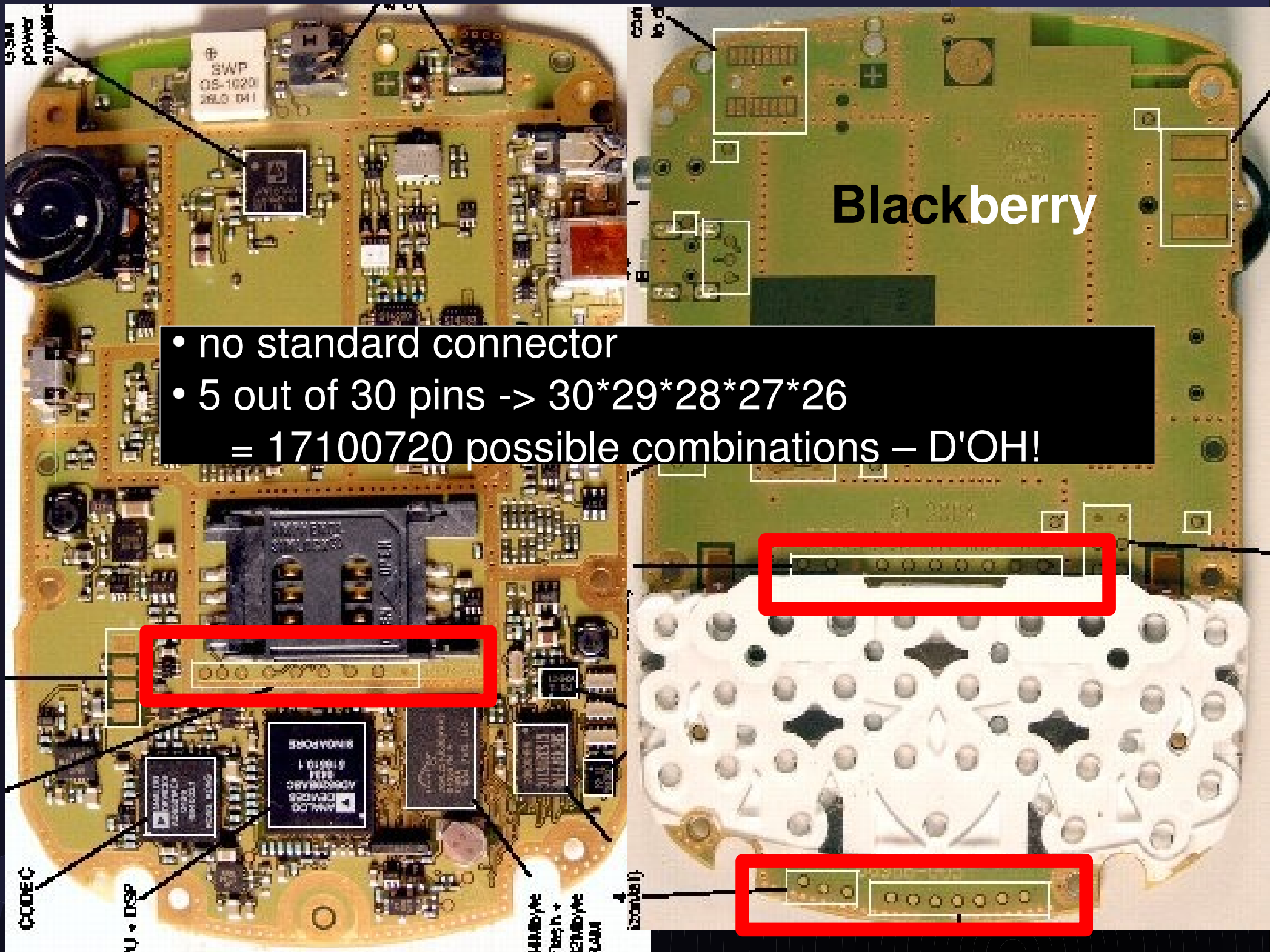
- often sufficient since
  - there are de-facto standard pinouts
  - IC pins are rare -> not so many testpins
- look for 1x5/6, 2x5, 2x7, 2x10 pin headers
- one of the rows (2x7 or 2x10) is GND for noise reduction
- often there are pullups (1-100k) for TMS, TDI and nTRST – nTRST could also be pulled low
- TDO should be high-impedance
- there might be a Vcc pin for the gluing logic
- be careful, you might fry your device!


D-Link Accesspoint
Testpins
D-Link
DL7500
8UA-ML06Y
0535-A-PB
D-Link
DL10050
FGK3479.1C
EON
EN29LV800BB-70TCP
14BN30A
0531KDA
ETRONTECH
EM636165TS-7
J331408HKF063.11
M-TEK
D-Link
DI-524
H/W Ver : B1
OK
03
25.000Y


D-Link Accesspoint – bottom side
one can see 4 lines from the test pins to the CPU


Testpins -
one for baseband,
one for CPU
ADSL Modem


Row of GND pins
ADSL Modem
Bottom side

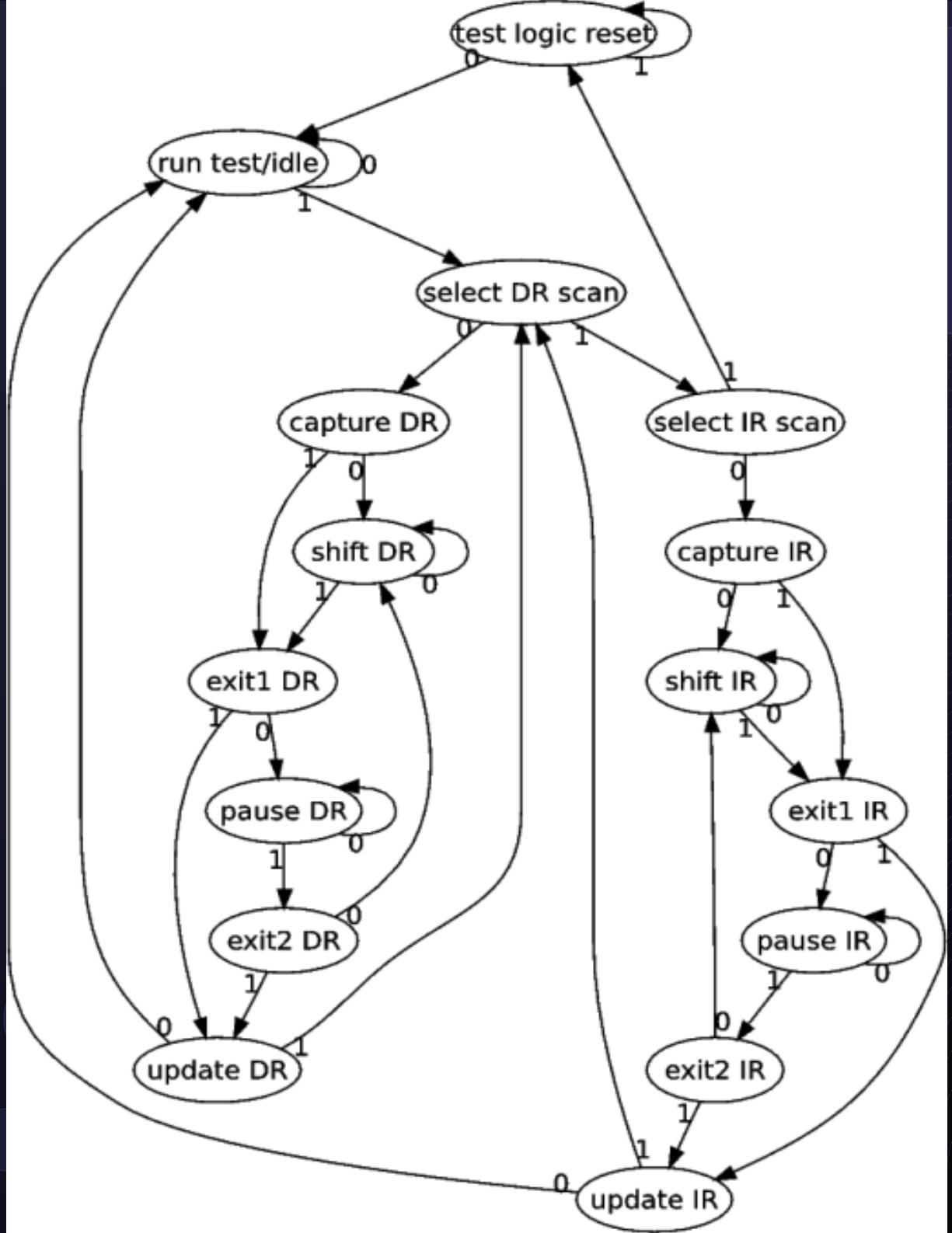

Blackberry
• no standard connector
• 5 out of 30 pins -> 30*29*28*27*26
= 17100720 possible combinations – D'OH!

How do we see that we got the right pins?

- at some time we will get output from TDO
- instead of including TDO in the pin-bruteforce, we monitor all IOs we're currently not driving for the TDO output
- now we got 4 out of 30 left for the bruteforce :-)

We don't need nTRST as well

- This way we can reduce the combinations to 3 out of 30
- Why don't we need it?
- Let's have a look at the state machine...



- TMS=11111 always leads to reset state!
- IR=Instruction Reg.
- DR=Data Reg.
- TDO is active in shift-IR/DR
- on TDO we get the Data we shifted in via TDI after Reg.length cycles
- thus we can shift in a pattern and look for it

Reducing the combinations to 2 out of 30?

- more exactly: (30*29)+28
- by searching (TCK,TMS) first and TDI then
- IR shall be loaded with „01“ - one could search for this pattern
- Problem: Some chips don't do that (Atmel AVR, ?)
- Additionally, a Bit-error could occur
- The chance that we get „01“ on some other active output while shifting is quite high (bad pattern)
- 3 out of 30 is good enough, we can shift our own identification-pattern to TDI and do some fuzzy-matching

Some problems left...

- a PC doesn't have 30 GPIOs
  - External (De-)Mux/Selection Logic at the parport? Too much fiddling, still software-shifting from PC...
  - Microcontrollers are cheap, flexible and can do the scanning
- detecting the pattern at TDO
  - best flexibility: PC-based detection (read all IOs every cycle)
  - but: 100MBytes of data for 30 IOs! -> USB or parport needed
  - or: Microcontroller does the detection – firmware can be modified for other pattern matching algorithms

Let's try not to fry our device

- Bruteforcing is dangerous if we drive a pin that's also driven by the device: short-circuit if 0 vs. 1
- We can't tell inputs and outputs apart for sure (might be driven suddenly while they were inputs before, etc.)
- If there's a pullup/down at an input, it's no longer high-impedance -> we'll think it's an output
- TDO is high-impedance while inactive
- simple solution: 330Ω resistors between our IOs and the device – at 3.3V that'll be harmeless 10mA max.
- just try any IO as output then

Current approach

- 8bit AVR ATmega16/32L microcontroller at 7.x MHz
- ~15€, serial port at 115200 baud, maybe usb2serial
- 30 IOs, max. <=500kHz, less than 1 hour for 30 pins

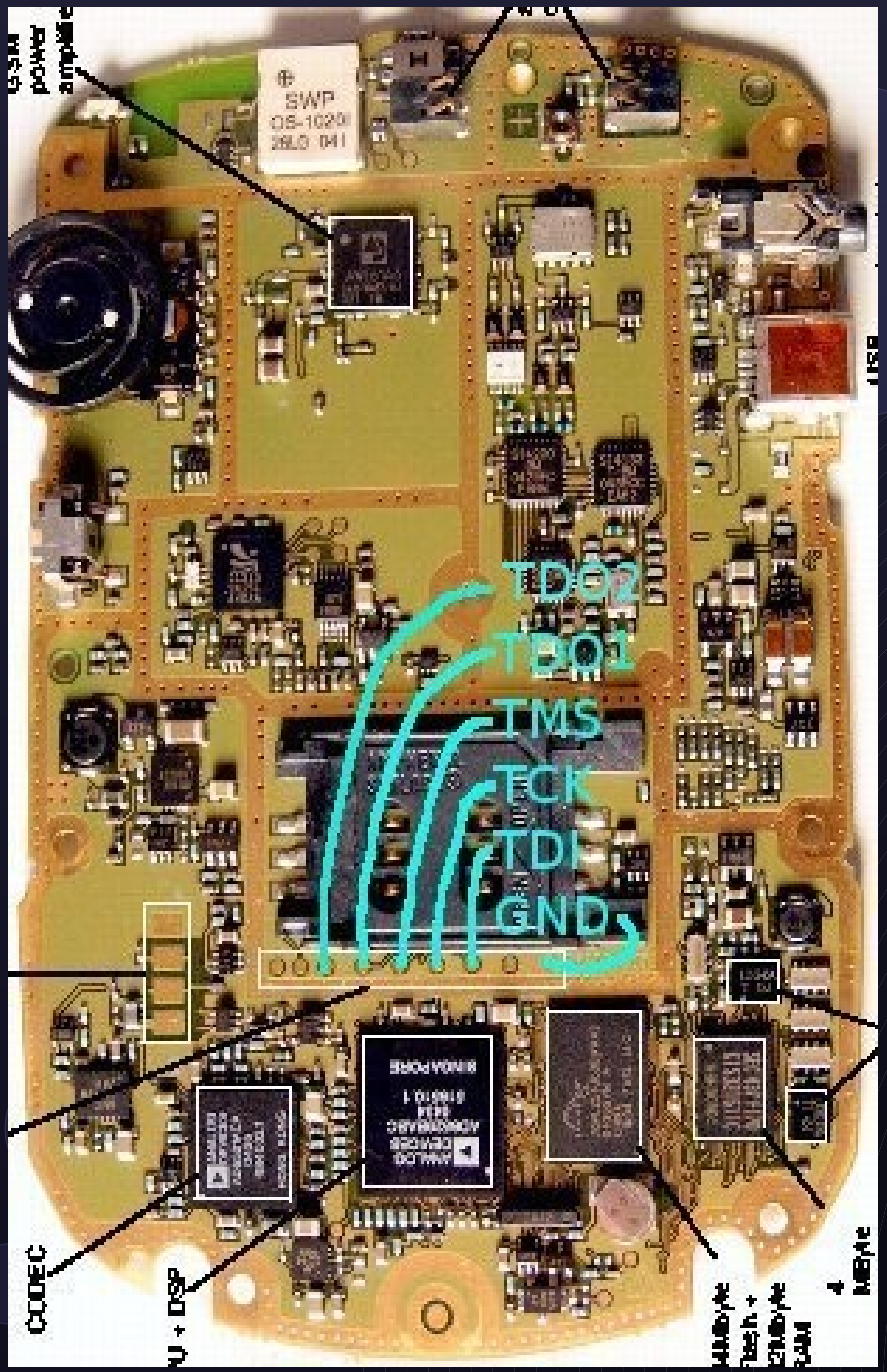
Pattern matching at TDO

- shift in a good pattern (0x23) several times (to get it through all IRs in the chain)
- count the matches for each pin
- look for pins with >=1 matches

If that doesn't work

- look for pins with a low level – they might be a pulled down nRST
  - try the bruteforce-game again while setting them to Hi one at a time
- try shifting the pattern more often
- try a weaker filter instead of the TDO pattern matching
  - for example you can dump the number of levelchanges for each pin while shifting
- JTAG might be disabled – see the wiki link from slide 1
- maybe there isn't a JTAG among the testpins at all
  - maybe there's a serial UART

Back to the Blackberry :-)



TDO1 has a smaller Instruction Register than TDO2.

I guess CPU and the codec are both in the chain you get at TDO2, while you only geht the CPU (or the codec?) at TDO1.

Haven't had a closer look at it (yet).

more details: C3A-Wiki